GraphQL

The GraphQL API allows queries to be made to Kibo's Storefront APIs in query syntax, allowing for mutations and modification of REST data. It is accessible directly by API call or through a Playground interface, along with a JavaScript client for handling authentication.

This guide describes how to install, configure, and authenticate a GraphQL client with code samples for example queries.

Access GraphQL

The GraphQL API can reached by appending `/graphql` to your Kibo Storefront home URL.

```
curl -X POST 'https://t00000.sandbox.mozu.com/graphql' \
  -H 'content-type: application/json' \
  -H 'Authorization: Bearer ' \
```

Include the following request headers:

- `Authorization` (required): Kibo API application auth token

The request body should include a JSON encoded body in the following form:

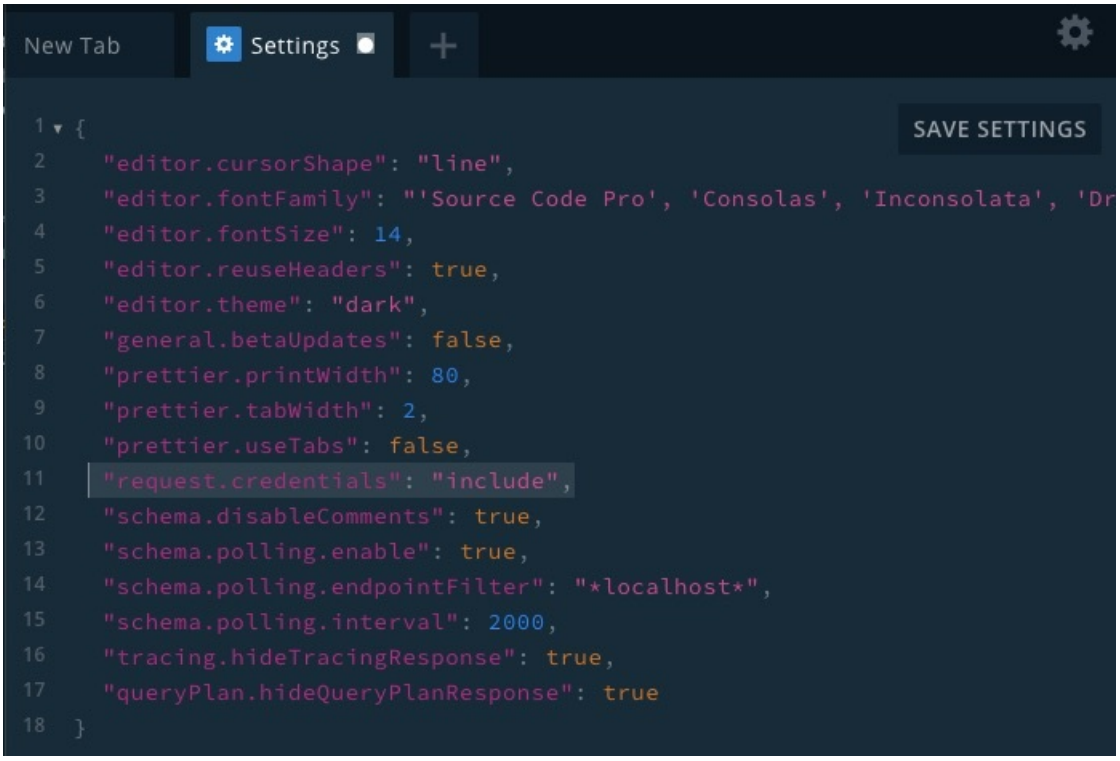
```
{
  "query": "...",
  "operationName": "...",
  "variables": { "myVariable": "someValue", ... }
}
```

- `query` (required): GraphQL query as a string
- `operationName` (optional): Only required if multiple operations are present in query
- `variables` (optional): Data to be passed to query

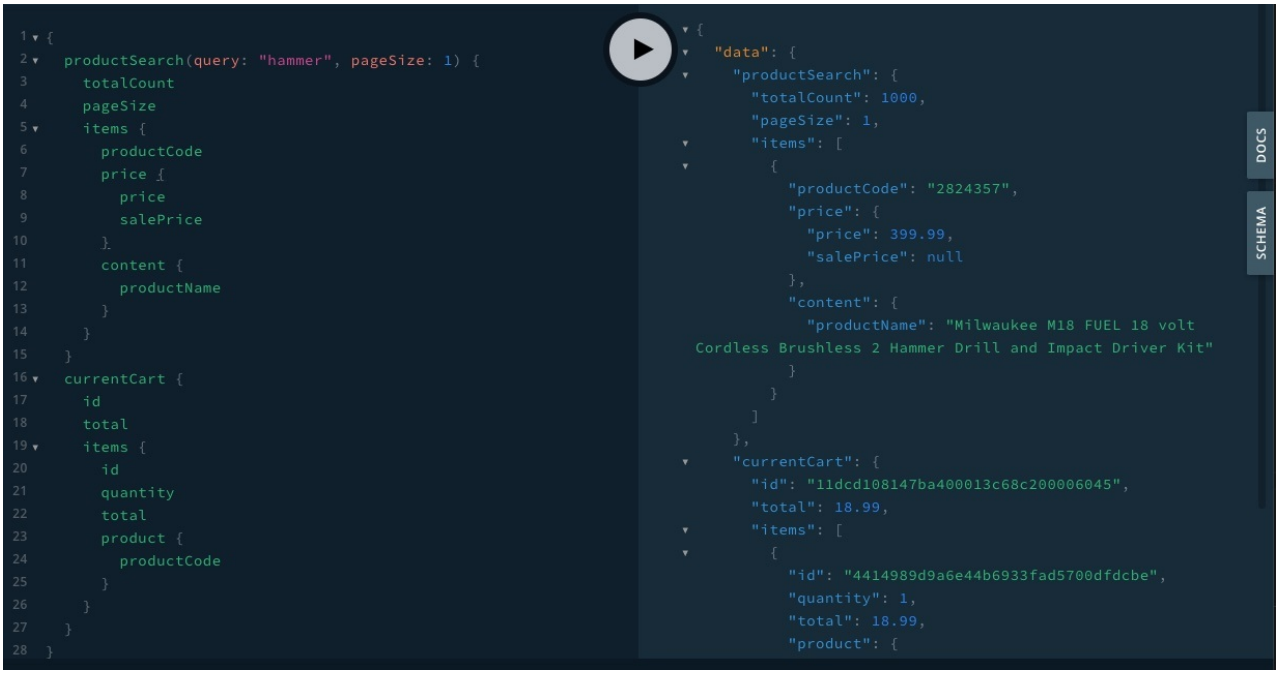
GraphQL Playground

The interactive GraphQL console, called the Playground, can be accessed from your site's front-end by appending `/graphql` to your storefront's homepage URL to access this tool in the browser.

You must first include your credentials (generated by the cookie from the front end site) by opening the gear icon in the top right corner and setting `"request.credentials": "include"` as shown below:



Then you will be able to build queries. GraphQL's syntax is displayed in the left pane of the Playground, while the JSON results will be displayed in the right pane once the call is made.



Code Samples

Get Product

```
query GetProduct {
  product(productCode: "1000165") {
    productCode
    content {
      productFullDescription
      productName
      productImages {
        imageUrl
      }
    }
    categories {
      content {
        name
      }
    }
    price {
      price
      salePrice
    }
  }
}
```

Get Categories

```
query GetCategories {
  categories(filter: "categoryCode eq shoes") {
    items {
      categoryCode
      content {
        name
        slug
        description
      }
    }
  }
}
```

Product Search

```
query Search {
  productSearch(query: "hammer") {
    totalCount
    items {
      productCode
      content {
        productName
      }
    }
    facets {
      label
      field
      values {
        label
        value
        isApplied
      }
    }
  }
}
```

Get Current Cart

* Requires a shopper access token to be sent as Bearer auth.

```
query GetCurrentCart {
  currentCart {
    total
    subtotal
    items {
      product {
        productCode
        name
      }
    }
  }
}
```

Get Cart By ID

```
query GetCart {
  cart(cartId: "12934ccc00db5100015d062b00007656") {
    total
    subtotal
    items {
      product {
        productCode
        name
      }
    }
  }
}
```

Add To Current Cart

* Requires a shopper access token to be sent in the header as Bearer auth.

```
mutation addToCart($addToCartInput:CartItemInput!) {
   addItemToCurrentCart(cartItemInput: $addToCartInput) {
  total
  id
  product {
       productCode
  }
   }
}
```

Variables:

```
{
 "addToCartInput": {
  "product": {
   "productCode": "SHOE-12"
  },
  "quantity": 1,
  "fulfillmentMethod": "Ship"
 }
}
```

Error Responses

GraphQL error messages include the following data.

- `message` : A detailed description of the error and root cause. This is always provided in the error response.
- `path` : A list of the query fields leading up to the error. This is only included when the error can be traced back to a particular field.
- `extensions` : Any additional information that may be included if applicable.
- `code` : An HTTP error code used by the Kibo APIs as documented here.

This information is provided in the below format:

```
{
 "errors": [
  {
   "message": "Description of the error.",
   "path": ["path"],
   "extensions": {
    "code": "Example"
   }
  }
 ]
}
```

GraphQL Client

There is a JavaScript Kibo GraphQL client available to handle API authentication and assist with registered and anonymous shopper authentication.

Prerequisites

If this client library is used in a browser environment, ensure your application is configured with storefront-only permissions as the API Key and secret will be visible.

Installation

To install the GraphQL client, run the following command:

```
npm install @kibocommerce/graphql-client
```

Configuration

To create an instance of the GraphQL client, the following configuration is required:

```
{
   "accessTokenUrl": "https://t00000.sandbox.mozu.com/api/platform/applications/authtickets/oaut
h",
   "clientId": "KIBO_APP.1.0.0.Release",
   "sharedSecret": "12345_Secret",
   "apiHost": "https://kibo-site.com"
}
```

- `apiHost` : Link to your GraphQL API instance.
- `accessTokenUrl` : Link to the Authentication Server, used to request an access token from Kibo's OAuth 2.0 service.
- `clientId` : Unique Application (Client) ID of your application.
- `sharedSecret` : Secret API key used to authenticate your application.

Based on the config, this integration will handle authenticating your application against the API using your Client ID and Shared Secret. These can be found from your Dev Center. For more information about the Dev Center and authentication, see Getting Started.

Authentication Hooks

Hooks can be provided to the `CreateApolloClient` method when creating a client that will execute on AuthTicket change, read, and remove events.

```
export interface KiboApolloClientConfig {
  api: KiboApolloApiConfig;
  clientAuthHooks: {
    onTicketChange: (authTicket: UserAuthTicket) => void;
    onTicketRead: () => UserAuthTicket;
    onTicketRemove: () => void;
  };
}
```

The auth ticket is passed as requests to the GraphQL server as a Bearer auth header, used to identify and authorize the user. This auth ticket works for both authenticated and anonymous shoppers, allowing for guest checkout scenarios.

These built-in hooks allow customization on storage methods for the user's auth ticket.

```
import { CreateApolloClient } from '@kibocommerce/graphql-client';

let currentTicket = getUserAuthorizationFromCooke();
const clientAuthHooks = {
    onTicketChange: (authTicket) => {
        currentTicket = authTicket;
        setUserAuthorizationCookie(authTicket);
    },
    onTicketRead: () => {
        return currentTicket;
    },
    onTicketRemove: () => {
        removeUserAuthorizationCookie();
    }
}

const client = CreateApolloClient({
    api: config,
    clientAuthHooks
});

const query = `query getCurrentCart {
    currentCart {
        total
    }
}`

const { data } = await client.query({ query });
```

Proxy Requests in Development

If you would like to see the requests to the GraphQL server, you need to set the `HTTPS_PROXY` environment variable to the proxy application you are using.

```
HTTPS_PROXY="http://127.0.0.1:8866"
```